GPU Accelerated Generalised Subclass Discriminant Analysis for Event and Concept Detection in Video

Stavros Arestis-Chartampilas
CERTH-ITI
Thermi 57001, Greece
stav_ares@iti.gr

Nikolaos Gkalelis
CERTH-ITI
Thermi 57001, Greece
gkalelis@iti.gr

Vasileios Mezaris
CERTH-ITI
Thermi 57001, Greece
bmezaris@iti.gr

ABSTRACT

In this paper a discriminant analysis (DA) technique called accelerated generalised subclass discriminant analysis (AGSDA) and its GPU implementation are presented. This method identifies a discriminant subspace of the input space in three steps: a) Gram matrix computation, b) eigenvalue decomposition of the between subclass factor matrix, and c) computation of the solution of a linear matrix system with symmetric positive semidefinite (SPSD) matrix of coefficients. Based on the fact that the computationally intensive parts of AGSDA, i.e. Gram matrix computation and identification of the SPSD linear matrix system solution, are highly parallelisable, a GPU implementation of AGSDA is proposed. Experimental results on large-scale datasets of TRECVID for event and concept detection show that our GPU-AGSDA method combined with LSVM outperforms LSVM alone in training time, memory consumption, and detection accuracy.

Categories and Subject Descriptors

H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing; I.2.10 [**Artificial Intelligence**]: Vision and Scene Understanding

Keywords

GPU; discriminant analysis; nonlinear; large-scale data.

1. INTRODUCTION

Nonlinear discriminant analysis (NDA) approaches [6], which aim at identifying a discriminant subspace of the original high-dimensional input space, and GPU implementations of computationally intensive algorithms, are recently getting increasing attention in large-scale video analysis problems [1, 2, 7, 11]. For instance, generalised subclass discriminant analysis (GSDA) combined with linear support vector machines (LSVMs) achieved excellent performance in multimedia event detection [7], while in [1, 2] GPU accelerated machine learning algorithms are presented, obtaining substantial speedups. Motivated by the above developments, a new DA method, called accelerated GSDA (AGSDA), and its GPU implementation are presented. The proposed GPU-AGSDA is evaluated on large-scale video datasets of TRECVID for the tasks of event and concept detection, providing promising performance.

The rest of the paper is structured as follows. In Sections 2 and 3, AGSDA and its GPU implementation are presented, respectively. Experimental results are reported in Section 4, and Section 5 presents conclusions and future work.

MM'15, October 26–30, 2015, Brisbane, Australia.

DOI: http://dx.doi.org/10.1145/2733373.2806321.

2. ACCELERATED GSDA

Subclass NDA techniques combined with linear classifiers have resulted in improved classification accuracy as well as lower storage requirements. The major drawback of these approaches is the computational cost for learning the DA transformation matrix. Recent approaches, such as GSDA [7], have decreased the training times significantly; however, further improvements are still necessary to deal with large-scale problems. In this section, exploiting the NDA framework proposed in [8], AGSDA is presented, achieving significant speedups in training time.

Let $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_N] = [\mathbf{X}_{1,1}, \ldots, \mathbf{X}_{\Omega,H_\Omega}] \in \mathbb{R}^{L\times N}$ be a subclass partition of an annotated training set of N observations $\mathbf{x}_r$, $r = 1, ..., N$, in the input space $\mathbb{R}^L$, where Ω is the number of classes, H_i is the number of subclasses of class i, $\mathbf{X}_{i,j} = [\mathbf{x}_{i,j}^1, \ldots, \mathbf{x}_{i,j}^{N_{i,j}}]$ is the subblock matrix containing the observations of the j-th subclass of class i, and $\mathbf{x}_{i,j}^n$, $N_{i,j}$ are the n-th observation and the number of observations of (i, j) subclass, respectively. Such a subclass partitioning can be easily created using an appropriate clustering algorithm, e.g. k-means. In order to deal with nonlinearly separable problems, a vector-valued function $\boldsymbol{\phi}(\cdot) : \mathbb{R}^L \to \mathbb{R}^F$, $\boldsymbol{\phi} = \boldsymbol{\phi}(\mathbf{x})$ may be used to map the partitioned training set $\mathbf{X}$ from the input space $\mathbb{R}^L$ to a higher- or infinite-dimensional space $\mathbb{R}^F$, where $\boldsymbol{\Phi} = [\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_N]$. It is also required that $\boldsymbol{\phi}(\cdot)$ is associated with a Mercer kernel evaluation function such that $\boldsymbol{\phi}_r^\top \boldsymbol{\phi}_q = k(\mathbf{x}_r, \mathbf{x}_q) = k_{rq}$. Assuming that in space $\mathbb{R}^F$ the data have zero mean, AGSDA seeks the column-orthogonal eigenvector matrix $\boldsymbol{\Gamma} \in \mathbb{R}^{N\times D}$ solving the following generalised eigenvalue problem (GEP) [7]

$$\mathbf{K}\mathbf{A}\mathbf{K}\boldsymbol{\Gamma} = \mathbf{K}\mathbf{K}\boldsymbol{\Gamma}\boldsymbol{\Delta}, \quad (1)$$

$$\mathbf{K} = \boldsymbol{\Phi}^\top\boldsymbol{\Phi}, \quad (2)$$

where $\mathbf{K}$ is the Gram matrix of the training set, $\boldsymbol{\Delta} \in \mathbb{R}^{D\times D}$ is the diagonal eigenvalue matrix with the real eigenvalues of the GEP in (1) along its diagonal ($D \ll F$). $\mathbf{A} \in \mathbb{R}^{N\times N}$

is the between subclass factor matrix, whose element $A_{r,q}$ corresponding to samples $\mathbf{x}_r \in \mathbf{X}_{i,j}$, $\mathbf{x}_q \in \mathbf{X}_{k,l}$ is defined as

$$A_{r,q} = \begin{cases} p_{i,j}(1-p_i)/(N_{i,j}N_{i,j}), & \text{if } (i,j)=(k,l), \\ 0 & \text{if } i=k, j\neq l, \\ -p_{i,j}p_{k,l}/(N_{i,j}N_{k,l}), & \text{otherwise,} \end{cases}$$

where p_i, $p_{i,j}$ are the estimated priors of the i-th class and (i,j) subclass respectively. Utilising the framework proposed in [8], the above problem may be solved in the following two steps: a) identifying the eigenpairs ($\mathbf{V}$,$\mathbf{\Lambda}$), $\mathbf{V} \in \mathbb{R}^{N\times D}$, $\mathbf{\Lambda} \in \mathbb{R}^{D\times D}$ of $\mathbf{A}$, b) Obtaining $\mathbf{\Gamma}$ by solving the following linear matrix system

$$\mathbf{K\Gamma} = \mathbf{V}. \tag{3}$$

The eigenvector matrix $\mathbf{V}$ in the first step of AGSDA can be efficiently computed following [8].

3. GPU ACCELERATION OF AGSDA

The most computationally intensive parts of AGSDA training are the computation of the Gram matrix (2) and the solution of the linear matrix system (3). The above computations are highly parallelisable. To this end, we employ GPU hardware to accelerate these parts, achieving a significant overall speedup, as explained below.

3.1 Gram Matrix Computation

The naive computation of the training Gram matrix (e.g. see (4) below for Gaussian RBF kernel) for large-scale data problems is associated with high memory and computational time requirements. To alleviate these, a tiled general matrix multiplication (GEMM) algorithm is used in this work, exploiting the GEMM function of CUDA cuBLAS library and the concurrency capabilities of recent GPU architectures. In order to gain insight into our implementation, we provide an example of computing the Gram matrix using the Gaussian RBF kernel

$$k_{rq} = k(\mathbf{x}_r, \mathbf{x}_q) = \exp(-\|\mathbf{x}_r - \mathbf{x}_q\|^2/2\sigma^2), \tag{4}$$

where k_{rq} is the element of $\mathbf{K}$ at the r-th row and q-th column. Directly evaluating (4) for each pair of training observations is very costly due to many additions/subtractions, which do not fully exploit the potential GPU parallelism, and are additionally more susceptible to precision loss. Furthermore, no function is included in CUDA 7.0 toolkit that can directly perform this operation.

In matrix form, the Gram matrix can be computed as

$$\mathbf{K} = \exp(\mathbf{D}), \qquad \mathbf{D} = -\frac{(\mathbf{B}+\mathbf{C}-2\mathbf{X}^\top\mathbf{X})}{2\sigma^2}, \tag{5}$$

where $\mathbf{C} = \mathbf{B}^\top$with the elements of matrix $\mathbf{B}$ defined as $b_{rq} = \mathbf{x}_r^\top\mathbf{x}_r$, $\forall\, r,q$, and $\mathbf{B},\mathbf{C} \in \mathbb{R}^{N\times N}$. Utilising the matrix form, $\mathbf{D}$ can be computed in the GPU with a GEMM call as follows

$$\mathbf{D} \;=\; \alpha\mathbf{X}^\top\mathbf{X} + \beta\mathbf{E},$$

where $\mathbf{E} = \mathbf{B}+\mathbf{C}$, $\alpha = \frac{1}{\sigma^2}$, $\beta = \frac{-1}{2\sigma^2}$. Using the GEMM function, a large parallelisation with a respective speedup is achieved. However, GPU memory size is still the main restriction of this formulation.

To overcome the above limitation as well, a tiled GEMM was implemented, where the involved matrices are appropriately partitioned. For demonstration purposes we consider a 1×2 partition for the data matrix

$$\mathbf{X} = [\mathbf{X}_1, \mathbf{X}_2], \tag{6}$$

where $\mathbf{X}_1 = [\mathbf{x}_1, \ldots, \mathbf{x}_{N'}]$, $\mathbf{X}_2 = [\mathbf{x}_{N'+1}, \ldots, \mathbf{x}_N] \in \mathbb{R}^{L\times N'}$, $N = 2N'$(and without loss of generality assuming that N is even). The overall problem is then formulated as

$$\begin{bmatrix} \mathbf{D}_{11} & \mathbf{D}_{12} \\ \mathbf{D}_{21} & \mathbf{D}_{22} \end{bmatrix} = \alpha \begin{bmatrix} \mathbf{X}_1^\top\mathbf{X}_1 & \mathbf{X}_1^\top\mathbf{X}_2 \\ \mathbf{X}_2^\top\mathbf{X}_1 & \mathbf{X}_2^\top\mathbf{X}_2 \end{bmatrix} + \beta \begin{bmatrix} \mathbf{E}_{11} & \mathbf{E}_{12} \\ \mathbf{E}_{21} & \mathbf{E}_{22} \end{bmatrix}.$$

Using the above fomulation and noting that $\mathbf{D}_{12} = \mathbf{D}_{21}^\top$, matrix $\mathbf{D}$ can be computed utilising three GEMM function calls as follows:

$$\mathbf{D}_{11} = \alpha\mathbf{X}_1^\top\mathbf{X}_1 + \beta\mathbf{E}_{11}, \tag{7}$$

$$\mathbf{D}_{21} = \alpha\mathbf{X}_2^\top\mathbf{X}_1 + \beta\mathbf{E}_{21}, \tag{8}$$

$$\mathbf{D}_{22} = \alpha\mathbf{X}_2^\top\mathbf{X}_2 + \beta\mathbf{E}_{22}. \tag{9}$$

The matrices involved in the above tiled GEMM function calls are considerably smaller in size compared to the full matrices in (5), thus alleviating the memory problem.

A further speedup can be achieved by exploiting the tiled GEMM procedure and properly pipelining GPU computations and CPU⇌GPU memory transactions (where ⇌ denotes a concurrent transaction), as shown in Fig. 1. In this figure, HD# and DH# denote Host-To-Device and Device-To-Host memory transactions respectively, K# denotes GPU computations, while CPU# denotes CPU computations. For instance, in our example, HD1 refers to the memory transactions to pass matrices $\mathbf{X}_1$ and $\mathbf{E}_{11}$ in the GPU RAM, K1 denotes the GEMM function (7), DH1 is the copying of the resulting matrix $\mathbf{D}_{11}$ back to CPU RAM and CPU1 denotes the calculation of $\mathbf{K}_{11}$ by exponentiating $\mathbf{D}_{11}$ according to the first part of (5). Using this notation for (7)-(9), and performing the pipelining described in Fig. 1, a 3–way concurrency is achieved in this example (and up to 4–way concurrency can be achieved in case of finer partitioning of the data matrix (6)).

We should also note that a formulation similar to the one described in this section can be used for computing the test-data kernel matrix, when applying the trained AGSDA method.

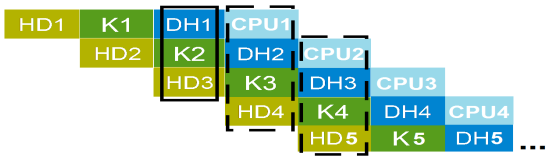

Figure 1: CPU⇌GPU concurrency.

3.2 Computation of Linear Matrix System Solution

Based on the fact that the Gram matrix is symmetric positive semi-definite (SPSD), the Cholesky factorisation is utilised to solve the linear matrix system in (3) for $\mathbf{\Gamma}$

$$\mathbf{K\Gamma} = \mathbf{V} \Rightarrow \mathbf{LL}^\top\mathbf{\Gamma} = \mathbf{V} \Rightarrow \begin{matrix} \mathbf{L\Psi} & = \mathbf{V}, \\ \mathbf{L}^\top\mathbf{\Gamma} & = \mathbf{\Psi}, \end{matrix} \tag{10}$$

where $\mathbf{L} \in \mathbb{R}^{N\times N}$ is a lower triangular matrix, and $\mathbf{\Psi} \in \mathbb{R}^{N\times D}$ is an unknown intermediate matrix. The built-in

dense linear solvers of CUDA 7.0 toolkit, namely cuSolver, are used. Specifically, the functions `cusolverDn<t>potrf()` and `cusolverDn<t>potrs()` are utilised to perform the Cholesky factorisation and the subsequent solution of the two linear matrix systems in (10) for $\mathbf{\Psi}$ and $\mathbf{\Gamma}$ respectively.

4. EXPERIMENTAL EVALUATION

In this section, GPU-AGSDA is evaluated in two different Nvidia graphic cards. Moreover, its performance is compared with the Matlab implementation of AGSDA and the C++ liblinear implementation of LSVM.

4.1 Datasets and Features

Three datasets are used for the experimental evaluation; specifically, we utilise subsets of the MED 2012 and SIN 2013 video corpora for event and concept detection:

MED-HBB: For event detection we used the publicly available partitioning of MED 2012 provided by the authors of [9]. It utilises a subset of the MED 2012 video corpus, comprising 25 target events and 13274 videos, and is divided to a training and evaluation set of 8824 and 4425 videos, respectively. We used improved dense trajectories (DT) to represent each video with a feature vector in $\mathbb{R}^{101376}$.

SIN13: For concept detection in video shots, 38 concepts of the SIN 2013 dataset are utilized. Two feature extraction procedures are used for representing the videos of the SIN dataset, as explained below:

a) *SIN13-LOCAL*: SIFT, SURF and ORB descriptors are applied to extract local features at keyframe-level. One keyframe per shot is used. The extracted features are encoded using VLAD, and compressed by utilizing a modification of the random projection matrix technique to provide a feature vector in $\mathbb{R}^{4000}$ for each keyframe [10].

b) *SIN13-CNN*: ConvNet network (CNN) features are employed for shot representation. Particularly, a 16-layer network [12] pre-trained using the 2009 ImageNet dataset [4] is employed. In our case, the last layer is utilized as output, providing a 1000-element vector representation for each keyframe. Again, one keyframe per shot is used.

4.2 Detection Approaches

We evaluated four different detection approaches. Specifically: GPU-AGSDA combined with LSVM running on two different Nvidia graphic cards, namely the low-end GeForce GTX 650 and the high-end Tesla K40; the Matlab implementation of AGSDA combined with LSVM; and the LSVM implementation of the C++ liblinear library [5]. In our AGSDA-based algorithms, the LSVM is implemented using the C++ libsvm library [3]. For convenience, we provide the following naming convention concerning the different algorithms, implementations and graphic cards used in the experiments:

a) AGSDA-1: GPU-AGSDA with LSVM, on GTX 650.
b) AGSDA-2: GPU-AGSDA with LSVM, on Tesla K40.
c) AGSDA-3: the Matlab version of AGSDA with LSVM.
d) LSVM: C++ liblinear implementation of LSVM, operating directly in the input space.

For all experiments we used Intel i7 3770K @3.5GHz, 32GB RAM machines with Win7 x64 OS. One set of parameters (Gaussian RBF parameter σ, LSVM parameter C) was estimated for each dataset using a grid search on a 3-fold cross-validation procedure, and the same parameter values were used for training all the detectors of the same detection approach. Moreover, the subclass parameters were set to $H_1 = 2, H_2 \in [1, 2]$, where H_1, H_2 refer to the number of subclasses for the target and rest-of-world classes, respectively.

4.3 Results

The detection results in terms of MXinfAP [13] for concept detection in SIN13-LOCAL and SIN13-CNN, and mean average precision (MAP) for event detection in MED-HBB for all approaches are shown in Fig. 2. In the same figure, the overall training and testing times along all concepts or events are reported. In Fig. 3 the distribution of the overall training and testing times of AGSDA-2 along its individual parts is shown. We observe the following:

- As shown in Fig. 2, the detection performance between our Matlab (AGSDA-3) and GPU implementation (AGSDA-1,-2) is equivalent. In addition, a 1.4% to 2.5% improvement of AGSDA+LSVM over LSVM is achieved. Overall, AGSDA-LSVM provides better detection results than LSVM for 30 out of 38 concepts in SIN13-CNN.

- Concerning training time, as shown in Fig. 2, GPU-AGSDA running on Tesla K40 (AGSDA-2) offers a significant speedup over both the LSVM of liblinear (which is a state-of-the-art LSVM implementation in terms of training time) and our Matlab implementation (AGSDA-3). Similarly, GPU-AGSDA running on GTX 650 (AGSDA-1) outperforms (MED-HBB, SIN13-CNN) or performs equivalently (SIN13-LOCAL) to our Matlab implementation. When compared to LSVM, it provides a 3× speedup in the MED-HBB dataset and an equivalent or slightly worse training time performance in the other two datasets.

- In Fig. 3, we observe that for the SIN13-LOCAL and SIN13-CNN datasets, the most computationally intensive parts of AGSDA-2 are the computation of the Gram matrix, the Cholesky factorisation and the LSVM training, while for MED-HBB dataset, the computation time is dominated by the memory transactions of the sorting algorithm, followed by the computations of the Gram matrix. Thus, for the latter dataset (MED-HBB) the provision of sorted observations (e.g. sorting the observations during the feature extraction procedure) can considerably decrease the overall computation time. We should also note that the LSVM training time can be considerably reduced using a faster LSVM library such as liblinear. However, the libsvm library was preferred due to its better robustness in relation to generalisation performance. We finally observed that in the testing stage, the computation of Gram representations of test observations is the bottleneck of our algorithm; this could be alleviated by using faster Gram representation algorithms [11].

Additionally, comparison with the results reported in [7] reveals a one to two orders of magnitude training time improvement over GSDA. Specifically, the learning time for one event using a 5×5 optimisation grid with GPU-AGSDA (on GTX 650) was approximately 5 minutes, while the reported time for the same experiment for GSDA [7] was above 50 minutes.

Further to the above, GPU-AGSDA also has lower memory requirements. Specifically, we observed that it required up to 8 times less memory than the corresponding Matlab implementation. This means, for instance, that on our hardware the GPU implementation avoided any use of HDD space as swap memory, in contrast to what the Matlab ver-

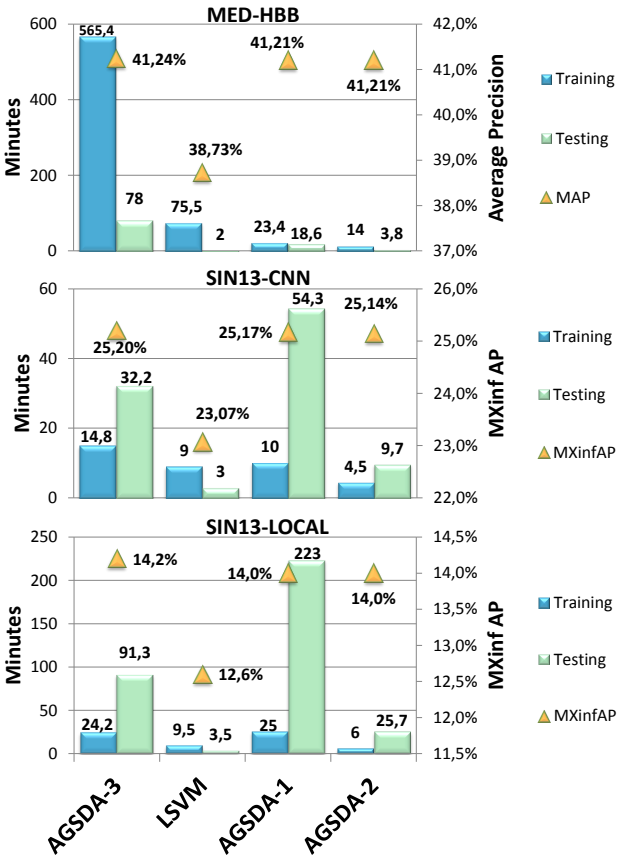

Figure 2: Experimental evaluation results.

sion of AGSDA did during the training part of the MED-HBB experiment.

5. CONCLUSIONS

In this paper a new efficient discriminant analysis method called accelerated generalized subclass discriminant analysis and its GPU implementation are presented. Specifically, a tiled GEMM procedure for the computation of the Gram matrix and the GPU implementation of Cholesky factorisation are exploited to further improve AGSDA in terms of training time and memory space requirements.

Future work directions include the investigation of a tiled linear matrix system solver, the exploitation of more efficient Gram matrix representation methods during the testing stage [11], and the porting of the proposed algorithm to Jetson TK1 embedded platform to achieve mobility and reduced power consumption.

6. ACKNOWLEDGEMENTS

This work was supported by the EC under contract FP7-600826 ForgetIT and by Nvidia corporation with the donation of a Tesla K40 GPU.

7. REFERENCES

[1] E. Agullo et al. Faster, cheaper, better – a hybridization methodology to develop linear algebra software for GPUs. *GPU Computing Gems*, 2, 2010.

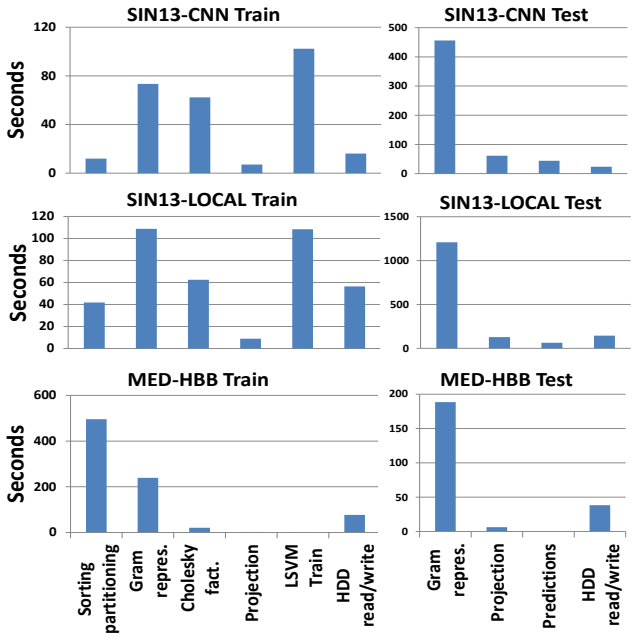

Figure 3: Bottlenecks of AGSDA on Tesla K40.

[2] A. Athanasopoulos et al. GPU acceleration for support vector machines. In *WIAMIS, Delft, NL, April 13-15*, 2011.

[3] C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.*, 2(3):27:1–27:27, May 2011.

[4] J. Deng et al. Imagenet: A large-scale hierarchical image database. In *IEEE CVPR*, pages 248–255, 2009.

[5] R.-E. Fan et al. LIBLINEAR: A library for large linear classification. *J. Mach. Learn. Res.*, 9:1871–1874, June 2008.

[6] N. Gkalelis et al. Mixture subclass discriminant analysis link to restricted Gaussian model and other generalizations. *IEEE Trans. on Neural Netw. and Learning Sys.*, 24(1):8–21, 2013.

[7] N. Gkalelis and V. Mezaris. Video event detection using generalized subclass discriminant analysis and linear support vector machines. In *ACM ICMR*, 2014.

[8] N. Gkalelis and V. Mezaris. Accelerated nonlinear discriminant analysis. *arXiv preprint arXiv:1504.07000*, 2015.

[9] A. Habibian et al. Recommendations for video event recognition using concept vocabularies. In *ACM ICMR*, pages 89–96, 2013.

[10] F. Markatopoulou et al. Local features and a two-layer stacking architecture for semantic concept detection in video. *IEEE Trans. on Emerging Topics in Computing*, 3(2):193–204, June 2015.

[11] B. Schölkopf and A. J. Smola. *Learning with kernels*. MIT Press, Cambridge, MA, USA, 2001.

[12] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[13] Yilmaz et al. A simple and efficient sampling method for estimating AP and NDCG. In *ACM SIGIR*, pages 603–610, NY, USA, 2008.